# Racket Programming Assignment #3: Lambda and Basic Lisp

Thomas Moskal

CSC 344 – Professor Graci

Due date: 3/7/22

## Learning Abstract:

This assignment is an exercise in the lambda function and properly using lists in racket programming. Lambda functions are a function that does not relate itself to a variable name. This way, you can quickly create functions in a program without tying up variable names. This makes it easier for a programmer who does not want to stop and create a completely new function. If you can learn when to properly use a Lambda expression you can save time and memory. The second task is practice with sets and basic lisp use. A good demonstration to reference if you have syntax questions. In the third task we altered the Sampler function. I really enjoyed getting to choose colors and consider what else a function like Sampler could be used for.

The final task of this assignment was a piece of code that emulates a deck of cards and can compare those cards in many ways. Eventually, the code determines the classification of a pair of cards in two-hand poker. After doing this assignment, I wondered what it would take to expand the function to take in a full hand of cards and a hand for the house. If I could achieve this, I would have a great program to teach poker noobs the value of the cards in their hand/in play. After floating this idea to an older friend who loves poker, I may have to end up completing such a program.

## Task 1 – Lambda

```
1a.
```

```
> ( ( lambda ( x ) ( list x (+ x 1 ) (+ x 2 ) ) ) 5 )
'(5 6 7)
> ( ( lambda ( x ) ( list x (+ x 1 ) (+ x 2 ) ) ) 0 )
'(0 1 2)
> ( ( lambda ( x ) ( list x (+ x 1 ) (+ x 2 ) ) ) 108 )
'(108 109 110)

1b.
> ( ( lambda ( x y z ) ( list z y x ) ) 'red 'yellow 'blue )
'(blue yellow red)
> ( ( lambda ( x y z ) ( list z y x ) ) 10 20 30 )
'(30 20 10)
> ( ( lambda ( x y z ) ( list z y x ) ) "Professor Plum" "Colonel Mustard" "Miss Scarlet")
'("Miss Scarlet" "Colonel Mustard" "Professor Plum")
```

```
1c.
```

```
> ( ( lambda ( x y ) (random x (+ y 1 ) ) ) 3 5 )
> ( ( lambda ( x y ) (random x (+ y 1 ) ) ) 3 5 )
3
> ( ( lambda ( x y ) (random x (+ y 1 ) ) ) 3 5 )
> ( ( lambda ( x y ) (random x (+ y 1 ) ) ) 3 5 )
> ( ( lambda ( x y ) (random x (+ y 1 ) ) ) 3 5 )
> ( ( lambda ( x y ) (random x (+ y 1 ) ) ) 3 5 )
> ( ( lambda ( x y ) (random x (+ y 1 ) ) ) 3 5 )
> ( ( lambda ( x y ) (random x (+ y 1 ) ) ) 3 5 )
> ( ( lambda ( x y ) (random x (+ y 1 ) ) ) 3 5 )
> ( ( lambda ( x y ) (random x (+ y 1 ) ) ) 3 5 )
> ( ( lambda ( x y ) (random x (+ y 1 ) ) ) 11 17 )
15
> ( ( lambda ( x y ) (random x (+ y 1 ) ) ) 11 17 )
> ( ( lambda ( x y ) (random x (+ y 1 ) ) ) 11 17 )
17
> ( ( lambda ( x y ) (random x (+ y 1 ) ) ) 11 17 )
> ( ( lambda ( x y ) (random x (+ y 1 ) ) ) 11 17 )
14
> ( ( lambda ( x y ) (random x (+ y 1 ) ) ) 11 17 )
> ( ( lambda ( x y ) (random x (+ y 1 ) ) ) 11 17 )
14
> ( ( lambda ( x y ) (random x (+ y 1 ) ) ) 11 17 )
> ( ( lambda ( x y ) (random x (+ y 1 ) ) ) 11 17 )
11
> ( ( lambda ( x y ) (random x (+ y 1 ) ) ) 11 17 )
14
>
```

## <u>Task 2 – List Processing Referencers and Constructors</u>

```
> ( define languages '(racket prolog haskell rust) )
> languages
'(racket prolog haskell rust)
> 'languages
'languages
> ( quote languages )
'languages
> ( car languages )
'racket
> (cdr languages)
'(prolog haskell rust)
> ( car ( cdr languages ) )
'prolog
> (cdr (cdr languages))
'(haskell rust)
> ( cadr languages )
'prolog
> (cddr languages)
'(haskell rust)
> (first languages)
'racket
> ( second languages )
'prolog
> (third languages)
'haskell
> ( list-ref languages 2 )
'haskell
> ( define numbers '(1 2 3 ) )
> ( define letters '(a b c ) )
> ( cons numbers letters )
'((1 2 3) a b c)
> ( list numbers letters )
'((1 2 3) (a b c))
> (append numbers letters)
'(1 2 3 a b c)
> ( define animals '(ant bat cat dot eel ) )
> ( car ( cdr ( cdr ( cdr animals ) ) ) )
'dot
> ( cadddr animals )
'dot
> ( list-ref animals 3 )
'dot
> ( define a 'apple )
> ( define b 'peach )
> ( define c 'cherry )
> ( cons a ( cons b ( cons c '() ) )
'(apple peach cherry)
> (listabc)
'(apple peach cherry)
> ( define x '(one fish) )
> ( define y '(two fish) )
> ( cons ( car x ) (cons (car (cdr x ) ) y ) )
'(one fish two fish)
> (append x y)
'(one fish two fish)
```

## <u>Task 3 – Little Color Interpreter</u>

```
> ( sampler )
(?): ( red orange yellow green blue indigo violet )
(?): ( red orange yellow green blue indigo violet )
indigo
(?): ( red orange yellow green blue indigo violet )
green
(?): ( red orange yellow green blue indigo violet )
(?): ( red orange yellow green blue indigo violet )
(?): ( red orange yellow green blue indigo violet )
orange
(?): ( aet ate eat eta tae tea )
(?): ( aet ate eat eta tae tea )
(?): ( aet ate eat eta tae tea )
                                                    3a.
(?): ( aet ate eat eta tae tea )
                                                   Sampler Demo
(?): ( aet ate eat eta tae tea )
                                                    \leftarrow
(?): ( aet ate eat eta tae tea )
(?): (0123456789)
(?): (0123456789)
(?): (0123456789)
(?): (0123456789)
(?): (0123456789)
(?): (0123456789)
(?): . user break
( define ( sampler )
  ( display "(?): " )
  ( define the-list ( read ) )
  ( define the-element
    ( list-ref the-list ( random ( length the-list ) ) ) )
  ( display the-element ) ( display "\n" )
  ( sampler )
                                                          Sampler Code
```

```
Welcome to DrRacket, version 8.3 [cs].
Language: racket, with debugging; memory limit 128 MB.

> (color-list)

? (random ("plum" "crimson" "indigo" "gold" "teal" "dark slate gray" "midnight blue"))

? (random ("plum" "crimson" "indigo" "gold" "teal" "dark slate gray" "midnight blue"))

? (random ("plum" "crimson" "indigo" "gold" "teal" "dark slate gray" "midnight blue"))

? (all ("plum" "crimson" "indigo" "gold" "teal" "dark slate gray" "midnight blue"))

? (2 ( "plum" "crimson" "indigo" "gold" "teal" "dark slate gray" "midnight blue"))

? (3 ( "plum" "crimson" "indigo" "gold" "teal" "dark slate gray" "midnight blue"))

2 (5 ( "plum" "crimson" "indigo" "gold" "teal" "dark slate gray" "midnight blue"))
```

## Color-list Demo

```
1 | #lang racket
3
    ( require 2htdp/image )
4
    :Makes the color bar
5
    ( define ( colored-bar color)
6
        ( rectangle 500 25 "solid" color)
8
 9
    ;Recursive helper function for "all" command
10
11
    (define (action-all list)
12
      (cond
13
        ((empty? list) (display "\n") )
14
        ( else
          (display (colored-bar (car list) ) )
15
           (display "\n")
16
          (action-all (cdr list) )
17
18
19
20
21
    ( define ( color-list )
22
       ( display "? " )
23
       ( define the-list ( read ) )
24
25
26
         ;Seperate the command from the first element of the original list
27
       ( define action (car the-list ) )
29
         ; cadr out the second element of the original list (the list of colors)
       ( define elements (cadr the-list ) )
30
31
32
       ( cond
33
          ; empty condition : Reread instructions late, didnt want to delete my code that error checks
34
          ((empty? elements )
35
            ( display "No elements input" ) )
          ;Random condition : Picks a random color from the list and displays it
36
37
          ((equal? action 'random )
38
             (displayIn ( colored-bar ( list-ref elements (random (length elements) ) ) ) ) )
39
          ;All condition : Displays the entire list in order (using recursion)
40
          (( equal? action 'all )
41
             ( action-all elements ) )
42
          ;Too high of an integer for command : Reread instructions late, didnt want to delete my code that error checks
43
          ( (> action (length elements) )
                (display "Not enough elements in set for this command.")
(display "The list only has ") (display (length elements))
(display "elements: ")
45
46
47
                ( display elements ) (display "\n") )
          ;Command is an integer : Specifically chooses and displays a color element
48
49
           ( (> action 0)
            (displayln ( colored-bar ( list-ref elements (- action 1) ) ) )
50
51
     (color-list)
52
```

## Color-list Code

## Task 4 – Two Card Poker

#### Playing cards Code -> 4a. Cards

#### Playing cards Demo

```
ranks 'X )
                                                                               ( ranks 'J )
                                                                      25
                                                                      26
                                                                               ( ranks '0 )
Welcome to DrRacket, version 8.3 [cs].
                                                                               ( ranks 'K )
Language: racket, with debugging; memory limit: 128 MB.
                                                                               ( ranks 'A )
> ( define cl '( 7 c ) )
                                                                      29
> ( define c2 '( Q H ) )
                                                                      30
                                                                      31
> c1
                                                                          ( define ( pick-a-card )
'(7 c)
                                                                      33
                                                                             ( define cards ( deck ) )
> c2
                                                                      34
                                                                             ( list-ref cards ( random ( length cards ) ) )
'(Q H)
                                                                      35
> ( rank cl )
                                                                      36
                                                                      37
                                                                          ( define ( show card )
                                                                      38
                                                                             ( display ( rank card ) )
> ( suit cl )
                                                                      39
                                                                             ( display ( suit card ) )
                                                                      40
> ( rank c2)
                                                                      41
'Q
                                                                          ( define ( rank card )
> (suit c2)
                                                                      43
                                                                             ( car card )
'Н
                                                                      44
                                                                      45
> (red? cl )
                                                                      46
                                                                          ( define ( suit card )
#f
                                                                      47
                                                                            ( cadr card )
> (red? c2)
                                                                      48
                                                                      49
> (black? cl)
                                                                          ( define ( red? card )
                                                                      50
#t
                                                                      52
                                                                               ( equal? ( suit card ) 'D )
> (black? c2)
                                                                      53
                                                                               ( equal? ( suit card ) 'H )
#f
                                                                      54
> (aces? '( A C ) '( A S ) )
                                                                      55
#t
                                                                      56
> (aces? '( K S ) '( A C ) )
                                                                      57
                                                                          ( define ( black? card )
#f
                                                                      58
                                                                             ( not ( red? card ) )
                                                                      59
> ( ranks 4 )
                                                                      60
'((4 C) (4 D) (4 H) (4 S))
                                                                      61
                                                                          ( define ( aces? cardl card2 )
> ( ranks 'K )
                                                                      62
                                                                             ( and
'((K C) (K D) (K H) (K S))
                                                                      63
                                                                               ( equal? ( rank cardl ) 'A )
                                                                               (equal? (rank card2) 'A)
> (length (deck) )
                                                                      64
> ( display (deck ) )
((2 C) (2 D) (2 H) (2 S) (3 C) (3 D) (3 H) (3 S) (4 C) (4 D) (4 H) (4 S) (5 C) (5 D) (5 H) (5 S)
(6 C) (6 D) (6 H) (6 S) (7 C) (7 D) (7 H) (7 S) (8 C) (8 D) (8 H) (8 S) (9 C) (9 D) (9 H) (9 S)
(X\ C)\ (X\ D)\ (X\ H)\ (X\ S)\ (J\ C)\ (J\ D)\ (J\ H)\ (J\ S)\ (Q\ C)\ (Q\ D)\ (Q\ H)\ (Q\ S)\ (K\ C)\ (K\ D)\ (K\ H)\ (K\ S)
(A C) (A D) (A H) (A S))
> ( pick-a-card )
'(5 D)
> ( pick-a-card )
'(6 H)
> ( pick-a-card )
'(5 C)
> ( pick-a-card )
'(6 H)
> ( pick-a-card )
'(X C)
> ( pick-a-card )
'(K S)
```

1.1

12 13

15

16

17

#lang racket

(require racket/trace ) ( define ( ranks rank ) ( list

( list rank 'C ) ( list rank 'D ) ( list rank 'H ) ( list rank 'S )

( define ( deck )

( ranks 2 )

( ranks 3 )

( ranks 6 ( ranks 7 ( ranks 8

( append

#### 4b. ur-classifier

```
Welcome to <u>DrRacket</u>, version 8.3 [cs].

Language: racket, with debugging; memory limit: 128 MB.

> ( pick-two-cards )
'((J C) (K H))
> ( pick-two-cards )
'((7 D) (9 D))
> ( pick-two-cards )
'((K D) (5 D))
> ( pick-two-cards )
'((9 S) (6 C))
> ( pick-two-cards )
'((8 D) (3 S))
```

#### Pick-two Demo

```
Welcome to DrRacket, version 8.3 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(5 H) '(K C))
< ' K
'K
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(3 S) '(6 D))
<6
6
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(4 H) '(9 D))
<9
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(X D) '(J H))
<'J
'J
> ( higher-rank ( pick-a-card ) ( pick-a-card ) )
>(higher-rank '(Q S) '(3 S))
<'0
'Q
```

### Higher rank demo

```
Welcome to DrRacket, version 8.3 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (classify-two-cards-ur ( pick-two-cards ) )
((6 D) (Q D)): Q high flush
> (classify-two-cards-ur ( pick-two-cards ) )
((X C) (Q S)): Q high
> (classify-two-cards-ur ( pick-two-cards ) )
((9 S) (6 S)): 9 high flush
> (classify-two-cards-ur ( pick-two-cards ) )
((J D) (7 D)): J high flush
> (classify-two-cards-ur ( pick-two-cards ) )
((A S) (2 C)): A high
> (classify-two-cards-ur ( pick-two-cards ) )
((8 H) (4 H)): 8 high flush
> (classify-two-cards-ur ( pick-two-cards ) )
((K C) (3 H)): K high
> (classify-two-cards-ur ( pick-two-cards ) )
((K D) (5 C)): K high
> (classify-two-cards-ur ( pick-two-cards ) )
((9 H) (7 S)): 9 high
> (classify-two-cards-ur ( pick-two-cards ) )
((9 H) (K H)): K high flush
> (classify-two-cards-ur ( pick-two-cards ) )
((9 S) (5 H)): 9 high
> (classify-two-cards-ur ( pick-two-cards ) )
((6 C) (J H)): J high
> (classify-two-cards-ur ( pick-two-cards ) )
((7 C) (X C)): X high flush
> (classify-two-cards-ur ( pick-two-cards ) )
((K H) (A S)): A high straight
> (classify-two-cards-ur ( pick-two-cards ) )
((X H) (8 S)): X high
> (classify-two-cards-ur ( pick-two-cards ) )
((K H) (J S)): K high
> (classify-two-cards-ur ( pick-two-cards ) )
((J C) (9 C)): J high flush
> (classify-two-cards-ur ( pick-two-cards ) )
((Q D) (5 D)): Q high flush
> (classify-two-cards-ur ( pick-two-cards ) )
((8 C) (K H)): K high
> (classify-two-cards-ur '(( J H )( J D )))
((J H) (J D)): J Pair
```

UR classifier demo

```
#lang racket
2
 3
    (require racket/trace )
 4
 5
    ( define ( ranks rank )
      ( list
 6
        ( list rank 'C )
        ( list rank 'D )
 8
 9
        ( list rank 'H )
        ( list rank 'S )
10
11
12
     )
13
    ( define ( deck )
14
15
      ( append
16
        ( ranks 2 )
        ( ranks 3 )
17
18
        ( ranks 4 )
19
        ( ranks 5 )
20
        ( ranks 6 )
21
        ( ranks 7 )
22
        ( ranks 8 )
23
         ( ranks 9 )
24
        ( ranks 'X )
25
        ( ranks 'J )
26
        ( ranks 'Q )
27
        ( ranks 'K )
         ( ranks 'A )
28
29
30
31
    ( define ( rank_conversion n )
32
33
      (cond
         ( (equal? n 'X ) 10)
34
35
         ( (equal? n 'J ) 11)
          ( (equal? n 'Q ) 12)
36
         ( (equal? n 'K ) 13)
37
38
          ( (equal? n 'A ) 14)
39
      )
40
41
42
    ( define ( convert_rank card)
43
      (cond
        (( number? (car card) )
44
45
          (car card) )
46
        (else
47
           (rank_conversion (car card))
48
49
50
51
52
    ( define ( pick-a-card )
53
      ( define cards ( deck ) )
       ( list-ref cards ( random ( length cards ) ) )
54
55
56
57
    ( define ( show card )
58
      ( display ( rank card ) )
59
      ( display ( suit card ) )
60
61
    ( define ( rank card )
62
     ( car card )
63
64
65
66
    ( define ( suit card )
67
      ( cadr card )
68
69
```

```
70 ( define ( red? card )
          ( equal? ( suit card ) 'D ) ( equal? ( suit card ) 'H )
 72
73
 74
75
 76
77
78
     79
 80
 81
     ( define ( aces? cardl card2 )
 82
        ( and
          ( equal? ( rank cardl ) 'A )
 83
          ( equal? ( rank card2 ) 'A )
 85
 86
 87
     ( define ( pick-two-cards )
 88
 89
        ( define cardl (pick-a-card) )
 90
         ( define card2 (pick-a-card) )
 91
        ( cond
           ((eq? (rank cardl) (rank card2))
 92
                    (pick-two-cards))
 94
           (else
                   (list cardl card2 ) )
 95
 96
 98
 99
     ( define ( higher-rank cardl card2 )
100
101
        ( cond
102
           ((equal? cardl card2)
103
             (higher-rank (pick-a-card) (pick-a-card) ) )
104
           (( > (convert_rank card1) (convert_rank card2) )
              ( car cardl ) )
105
106
           (else
107
              ( car card2 )
108
109
110
111
112
     ( define (classify-two-cards-ur ulti-list )
        (display ulti-list )
(display ": ")
(define cardl (car ulti-list) )
113
114
115
116
         (define card2 (cadr ulti-list) )
117
         (define high ( higher-rank cardl card2 ) )
118
         ( display high )
119
        (cond
120
          ((equal? (cadr cardl) (cadr card2) )
121
             ((or (= (- ( convert_rank card1 ) ( convert_rank card2 ) ) 1) (= (- ( convert_rank card2 ) ( convert_rank card1 ) ) 1) ) ( display " high straight flush" ) )
122
123
124
125
              (display " high flush" ) ) )
126
        (else
127
         (cond
           128
129
130
131
            (cond
              ((equal? (car cardl) (car card2) )
(display " Pair" ) )
132
133
134
135
                (display " high" ) ) ) ) )
136
137
    ;(classify-two-cards-ur ( pick-two-cards ) )
;(classify-two-cards-ur '(( J H )( J D )))
;( trace higher-rank )
139
140
```

#### UR classifier code

#### 4c. classifier

```
Welcome to DrRacket, version 8.3 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (classify-two-cards ( pick-two-cards ) )
((9 H) (6 S)): Nine high
> (classify-two-cards ( pick-two-cards ) )
((5 C) (7 S)): Seven high
> (classify-two-cards ( pick-two-cards ) )
((4 H) (K D)): King high
> (classify-two-cards ( pick-two-cards ) )
((Q H) (6 S)): Queen high
> (classify-two-cards ( pick-two-cards ) )
((8 S) (3 C)): Eight high
> (classify-two-cards ( pick-two-cards ) )
((6 S) (7 D)): Seven high straight
> (classify-two-cards ( pick-two-cards ) )
((6 C) (3 D)): Six high
> (classify-two-cards ( pick-two-cards ) )
((A D) (3 D)): Ace high flush
> (classify-two-cards ( pick-two-cards ) )
((7 D) (3 D)): Seven high flush
> (classify-two-cards ( pick-two-cards ) )
((Q H) (4 C)): Queen high
> (classify-two-cards ( pick-two-cards ) )
((9 H) (3 H)): Nine high flush
> (classify-two-cards ( pick-two-cards ) )
((3 C) (A C)): Ace high flush
> (classify-two-cards ( pick-two-cards ) )
((3 S) (9 C)): Nine high
> (classify-two-cards ( pick-two-cards ) )
((Q C) (K C)): King high straight flush
> (classify-two-cards ( pick-two-cards ) )
((A H) (8 S)): Ace high
> (classify-two-cards ( pick-two-cards ) )
((A D) (Q H)): Ace high
> (classify-two-cards ( pick-two-cards ) )
((9 H) (8 S)): Nine high straight
> (classify-two-cards ( pick-two-cards ) )
((A C) (2 H)): Ace high
> (classify-two-cards ( pick-two-cards ) )
((6 C) (8 H)): Eight high
> (classify-two-cards ( pick-two-cards ) )
((9 H) (8 C)): Nine high straight
```

#### Classifier Demo

```
1 | #lang racket
    (require racket/trace )
 3
 4
    ( define ( ranks rank )
 5
 6
       ( list
         ( list rank 'C )
 7
         ( list rank 'D )
 8
         ( list rank 'H )
9
         ( list rank 'S )
10
11
12
13
    ( define ( deck )
15
       ( append
16
         ( ranks 2 )
17
         ( ranks 3 )
18
         ( ranks 4 )
         (ranks 5)
19
20
         ( ranks 6 )
         ( ranks 7 )
21
22
         ( ranks 8 )
         ( ranks 9 )
23
         ( ranks 'X )
24
         ( ranks 'J )
25
         ( ranks 'Q )
26
27
         ( ranks 'K )
28
         ( ranks 'A )
29
30
31
    ( define ( rank_conversion n )
33
       (cond
          ( (equal? n 'X ) 10)
34
          ( (equal? n 'J ) 11)
35
          ( (equal? n 'Q ) 12)
36
          ( (equal? n 'K ) 13)
37
          ( (equal? n 'A ) 14)
38
39
40
41
    ( define ( english_conversion n )
42
43
       (cond
44
         ( (equal? n 2 ) "Two")
         ( (equal? n 3 ) "Three")
45
         ( (equal? n 4 ) "Four")
47
         ( (equal? n 5 ) "Five")
         ( (equal? n 6 ) "Six")
49
         ( (equal? n 7 ) "Seven")
         ( (equal? n 8 ) "Eight")
50
51
         ( (equal? n 9 ) "Nine")
          ( (equal? n 'X ) "Ten")
52
          ( (equal? n 'J ) "Jack")
53
          ( (equal? n 'Q ) "Queen")
54
          ( (equal? n 'K ) "King")
( (equal? n 'A ) "Ace")
55
56
57
      )
58
59
60
61
    ( define ( convert_rank card)
62
      (cond
63
        (( number? (car card) )
64
           (car card) )
65
         (else
66
            (rank conversion (car card))
67
68
      )
69
70
71
    ( define ( pick-a-card )
       ( define cards ( deck ) )
72
       ( list-ref cards ( random ( length cards ) ) )
73
74
75
76
    ( define ( show card )
77
       ( display ( rank card ) )
78
       ( display ( suit card ) )
79
```

```
81 ( define ( rank card )
 82
        ( car card )
 83
 84
      ( define ( suit card )
 85
        ( cadr card )
 86
 87
 88
 89
     ( define ( red? card )
 90
         ( or
           ( equal? ( suit card ) 'D )
 91
           (equal? (suit card) 'H)
 92
 93
 94
 95
 96
      ( define ( black? card )
 97
        ( not ( red? card ) )
 98
 99
      ( define ( aces? cardl card2 )
100
101
         ( and
           ( equal? ( rank cardl ) 'A ) ( equal? ( rank card2 ) 'A )
102
103
104
105
106
107
      ( define ( pick-two-cards )
         ( define cardl (pick-a-card) )
( define card2 (pick-a-card) )
108
109
110
         ( cond
111
            ((eq? (rank cardl) (rank card2))
112
                      (pick-two-cards))
113
            (else
114
                     (list cardl card2 ) )
115
116
117
     ( define ( higher-rank cardl card2 )
118
119
120
121
            ((equal? cardl card2)
            (higher-rank (pick-a-card) (pick-a-card) ) ) (( > (convert_rank cardl) (convert_rank card2) )
122
123
124
                ( car cardl ) )
125
            (else
126
               ( car card2 )
            )
127
128
129
130
131
      ( define (classify-two-cards ulti-list )
         (display ulti-list )
(display ": ")
132
133
134
         (define cardl (car ulti-list) )
135
          (define card2 (cadr ulti-list) )
136
         (define high ( higher-rank cardl card2 ) )
         (define english_name ( english_conversion high ) ) (display english_name)
137
138
139
140
           ((equal? (cadr cardl) (cadr card2) )
141
             (cond
              ((or (= (- ( convert_rank cardl ) ( convert_rank card2 ) ) l) (= (- ( convert_rank card2 ) ( convert_rank cardl ) ) l) )
      ( display " high straight flush" ) )
142
143
144
145
                (display " high flush" ) ) )
146
         (else
147
          (cond
            ((or (= (- ( convert_rank card1 ) ( convert_rank card2 ) ) 1) (= (- ( convert_rank card2 ) ( convert_rank card1 ) ) 1) )
149
                 (display " high straight" ))
            (else
150
151
             (cond
                ((equal? (car cardl) (car card2) )
152
153
                 (display " Pair" ) )
154
                (else
                 (display " high" ) ) ) ) )
155
156
157
     ;(classify-two-cards ( pick-two-cards ) );(classify-two-cards '(( A H )( A D )))
158
159
160
     ; ( trace higher-rank )
```

#### Classifier Code